Yakup Gorur



**1. Write MATLAB/Python routines.to perform following operations.**

a. Read original.jpg color image file, convert it to 8bits gray level image and write the gray level image to original_gray.jpg file.

original_gray.jpg 48 KB

original.jpg 58 KB

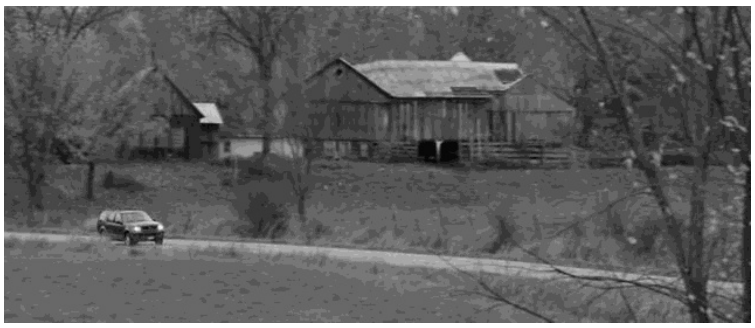
b. Read original_gray.jpg image file and calculate its 16bin histogram and plot the histogram

c. While you are calculating the 16bin histogram you quantize the 8bits gray level image to 4bits gray level image. Write the quantized 4 bits gray level image to original_gray_4.jpg file.

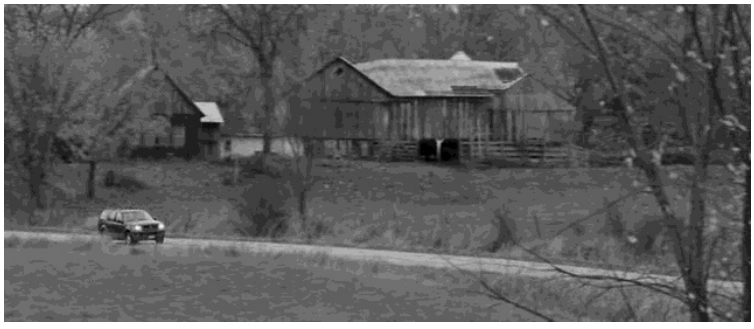
d.Check size of original_gray.jpeg and original_gray_4.jpg images and comment on it.

**16bin Histogram** **256bin Histogram**

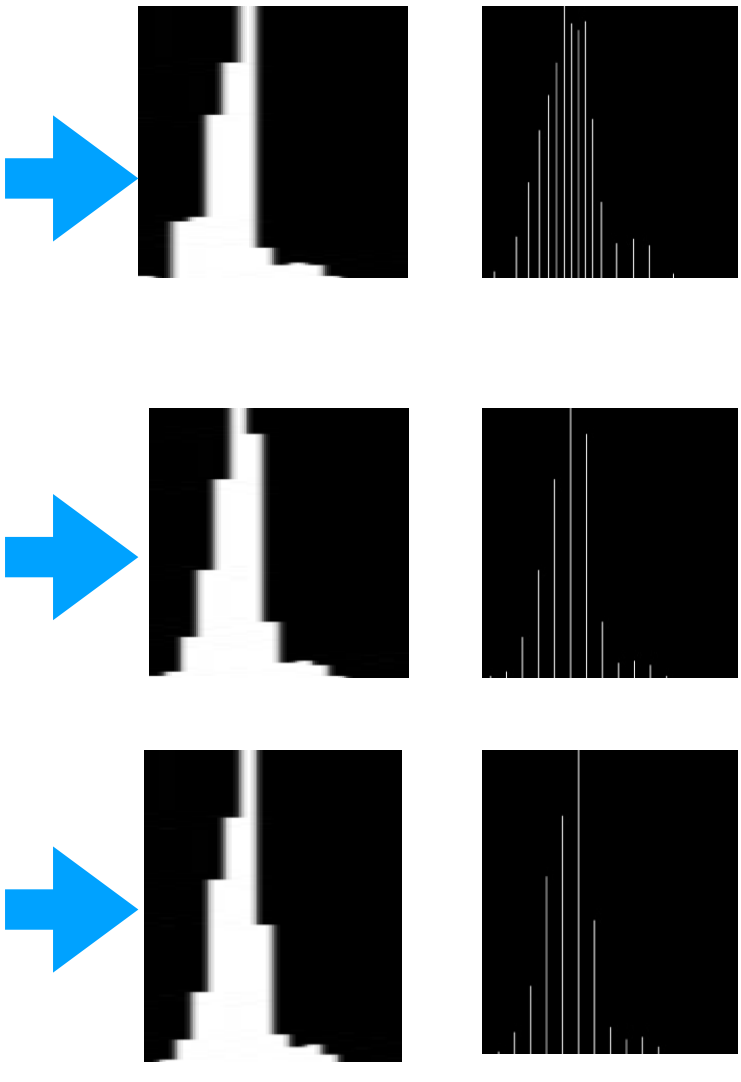
original_gray_4_Kmeans.jpg 60 KB



original_gray_4_lut.jpg 70 KB



original_gray_4_withbasic.jpg 72 KB



- After the quantization, the image file size increase. Although there are more colour than quantizeted image in the original gray image, the quantized image file size higher than original. The jpeg image format includes this 3 steps:
- Color space definition (same)
- Component sub-sampling registration
- Pixel aspect ratio definition. (same)

In the component sub-sampling registration process, The quantized image file can have been size because of the Compression Artifacts:

Saving an image with JPEG compression adds artifacts to the file. Even if they are not visible in the first place, the image data is different. Compressing an image with artifacts again, generates more artifacts and so on.

In this case, JPEG compression is not as effective as with the original file. The more artifacts are contained, the bigger the file will be.

**2. Write a MATLAB/Python routine.to perform histogram equalization.**

a. Read distorted.jpg image file, convert it to 8bits gray level image and write the gray level image to distorted_gray.jpg file.

distorted.jpg

distorted_gray.jpg

b. Apply histogram equalization on distorted_gray.jpg and write the output image on histeq_distorted_gray.jpg file.

histeq_distorted_gray.jpg

c. Write the algorithmic steps of your histogram equalization algorithm.

Histogram equalization (EqualizeHist_Function):
Step 1: Calculate the histogram of image.
Step 2: Calculate the Cumulative Histogram from histogram which is calculated Step 1.
Step 3: Divide the Cumulative Histogram value with number of pixel in the image and find Cumulative Density Function (CDF)
Step 4: Multiply the CDF values with (Gray levels -1 )
(in our case: 8 bit image file -> the Gray Level = 256, so multiply with 255)
Step 5 (final): map the new gray level values into number of pixels.

| Gray Level Value | CDF | CDF * (Levels-1) |
|---|---|---|
| 0 | 0.11 | 0 |
| 1 | 0.22 | 1 |
| 2 | 0.55 | 3 |
| 3 | 0.66 | 4 |
| 4 | 0.77 | 5 |
| 5 | 0.88 | 6 |
| 6 | 0.99 | 6 |
| 7 | 1 | 7 |

Step 4

(frequency)
Assume old gray levels values has these number of pixels

| Gray Level Value | New Gray Level Value | Frequency |
|---|---|---|
| 0 | 0 | 2 |
| 1 | 1 | 4 |
| 2 | 3 | 6 |
| 3 | 4 | 8 |
| 4 | 5 | 10 |
| 5 | 6 | 12 |
| 6 | 6 | 14 |
| 7 | 7 | 16 |

Step 5

**3. Write a MATLAB/Python routine.to perform histogram matching.**

a. Apply histogram matching on distorted_gray.jpg when the desired image file is original_gray.jpg. Write the output image on histmatch_distorted_gray.jpg file.

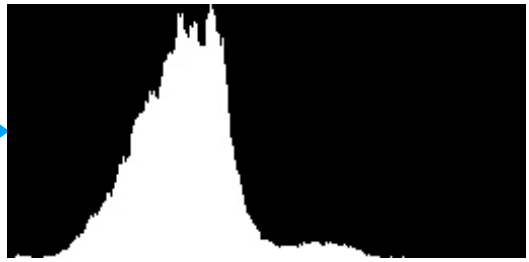
b. Write the algorithmic steps of your histogram matching algorithm.

c. Calculate the gray level histograms of the image files, original_gray.jpg, histeq_distorted_gray.jpg and histmatch_distorted_gray.jpg. Plot the histograms.

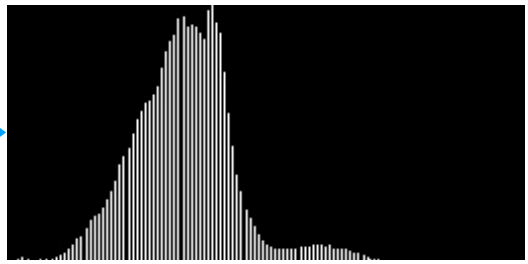
d. Compare the plotted histograms and the corresponding gray level images. Write your comments.

**Image** **Histogram**

distorted_gray.jpg 36 KB



original_gray.jpg 48 KB



histmatch_distorted_gray.jpg 47 KB

72 KB

**Histogram Matching Algorithm (HistogramMatching):**

- Input -> inputImage, desiredImage
- Output -> outputImage

- **Calculate the Histogram and Cumulative Histogram of Input Image.**

  Cumulative Histogram Of Input Image = $H_i$
- **Calculate the Histogram and Cumulative Histogram of Desired Image**

  Cumulative Histogram Of desired Image = $H_d$
- **Histogram Matching Algorithm:**

*For each intensity in the original image, find an intensity in the transformed image that has as close as possible, the same amount of Cumulative Histogram.*

Specifically, for each input level "$x$", find an output level "$y$" so that $H_d[y]$ best matches $H_i[x]$:

$$|H_i[x] - H_d[y]| = min_z |H_i[x] - H_d[z]|$$

and then I setup a lookup entry: $lookup[x] = y$

The result of histogram equalization nearly quite similar to original image. As seen their histogram graphics, the patterns are similar between original and output images. It can be said that, although the distorted image is corrupted, it does not loss so much information. It can be returned with histogram equalization.

**The written function:**

```
//quantize function to reduce bits
void Quantize_Function(const cv::Mat &input, cv::Mat &output, size_t div);
void Quantize_Function_with_K_Means(const cv::Mat &input, cv::Mat &output, int K);

//Histogram Equalization
void EqualizeHist_Function(const Mat1b& src, Mat1b& dst);

//Calculate the Histogram
void Image2Histogram(Mat image, float histogram[]);

//Calculate the Cumulative Histogram
void Histogram2CumulativeHistogram(float histogram[], float cumulativeHistogram[]);

//Histogram Matching
void HistogramMatching(const Mat& inputImage, const Mat& desiredImage, Mat&
outputImage);

//Show Histogram
void showHistogram(const Mat& image, string fileName);
void showHistogram16bin(const Mat& image, string fileName);

int main(int argc, const char * argv[])
```

This project prepared with C++ programming language and OpenCV library.
Full Code available at: https://github.com/ykpgrr/Some-Histogram-Algorithms